Freezing Simulates Non-Freezing Tile Automata

Cameron Chalk[1], Austin Luchsinger[2*], Eric Martinez[2], Robert Schweller[2*],
Andrew Winslow[2], and Tim Wylie[2*]

[1] Department of Electrical and Computer Engineering, University of Texas - Austin
ctchalk@utexas.edu
[2] Department of Computer Science, University of Texas - Rio Grande Valley
{austin.luchsinger01,robert.schweller,andrew.winslow,timothy.wylie}@utrgv.edu

Abstract. Self-assembly is the process by which a system of particles randomly agitate and combine, through local interactions, to form larger complex structures. In this work, we fuse a particular well-studied generalization of tile assembly (the 2-*Handed* or *Hierarchical Tile Assembly Model*) with concepts from cellular automata such as states and state transitions characterized by neighboring states. This allows for a simplification of the concepts from active self-assembly, and gives us machinery to relate the disparate existing models. We show that this model, coined *Tile Automata*, is invariant with respect to *freezing* and *non-freezing* transition rules via a simulation theorem showing that any non-freezing tile automata system can be simulated by a freezing one. Freezing tile automata systems restrict state transitions such that each tile may visit a state only once, i.e., a tile may undergo only a finite number of transitions. We conjecture that this result can be used to show that the *Signal-passing Tile Assembly Model* is also invariant to this constraint via a series of simulation results between that model and the Tile Automata model. Further, we conjecture that this model can be used to consolidate the several oft-studied models of self-assembly wherein assemblies may break apart, such as the Signal-passing Tile Assembly Model, the *negative-glue* 2-Handed Tile Assembly Model, and the *Size-Dependent Tile Assembly Model*. Lastly, the Tile Automata model may prove useful in combining results in cellular automata with self-assembly.

1 Introduction

A diverse collection of different algorithmic self-assembly models have emerged in recent years to explore the theoretical power of self-assembling systems under a wide variety of experimentally motivated constraints. While many important results continue to develop within these models, relatively little is known about how models which allow active self assembly and/or disassembly relate to each other. In this paper we propose to develop a tool set for proving connections between a large set of diverse self-assembly models. Our approach is based on the proposal of a new mathematical abstraction we term *Tile Automata* (TA) which combines elements of passive tile self-assembly (such as the 2HAM [2])

with local state change rules similar to asynchronous cellular automata (see [9] for a survey on cellular automata). Our goal is to study fundamental properties of active self-assembly and connect the disparate models with this new abstract model and a powerful tool set.

Active Self-Assembly. Self-assembly is the process by which a system of particles randomly agitate and combine through local interactions to form larger and more complex structures. Many forms of self-assembly are passive in nature, meaning the component system monomers are static with no internal changing of state, and simply interact based on a fixed surface chemistry. Newer models of self-assembly add an *active* component where system particles store an internal state that may adjust based on local interactions. These state changes affect how a particle interacts with others. Active self-assembly models may include substantial power (such as movement [4,12]) with an eye toward future technologies and swarm robotics. Other models focus on experimental techniques within emerging technologies such as DNA strand displacement cascades which permit a form of *signal passing* within tile systems [8,10,11].

Tile Automata. Tile Automata components are stateful square tiles living in a 2D grid. Pairs of states may be assigned an affinity value, allowing assembled collections of tiles to combine if a required threshold of affinity between the two assemblies is reached. In this way, Tile Automata incorporates 2-handed self-assembly. Similar to asynchronous cellular automata, a collection of transition rules dictate state changes based on local neighbor states. Thus, tiles within an assembly may undergo state changes, altering the internal affinities by which the assembly is bound. If new affinities are added, new combination events may occur. If affinities are removed, previously stable assemblies may become unstable and break apart.

The Tile Automata model is similar and partially inspired by the *nubots* model [12]. However, an important limitation with Tile Automata is the absence of a *movement* rule, which is a key feature prominent in nubot literature. Instead, Tile Automata is closely linked to models such as the signal tile model [10], and the active self-assembly model [8], in which tile self-assembly is augmented with a signal passing scheme permitting glues on tile edges to flip on and off dynamically. The Tile Automata model attempts to abstract away some of the specifics of these models to allow for a cleaner mathematical approach to understanding fundamental capabilities within this type of active self-assembly.

Our Contribution. Our primary result in this work is proving that *freezing*[3] Tile Automata systems, in which a tile must never revisit the same state twice, can simulate *non-freezing* systems, which have no such restriction. This shows that within self-assembly, freezing and non-freezing systems are equiv-

---

[3] We borrow the notion of freezing from the cellular automata literature [1,6,7]. There are two informal perspectives towards freezing that are equivalent in CA but not equivalent in TA. One is that a cell (tile) must never revisit the same state twice. The other is that a position in $\mathbb{Z}^2$ must never revisit the same state twice. Intuitively, in TA, a position may see several tiles due to tiles attaching and detaching. Thus, the perspectives are different. We choose the first perspective, matching the notion that tiles themselves are stateful, and positions in space are not stateful.

alent up to constant scale simulation. This is in contrast to freezing within cellular automata [6,7], in which freezing systems are substantially weaker than non-freezing. The intuition for this contrast is that cellular automata cells are "stuck" in place, and thus "frozen" cells become useless, whereas "frozen" tiles may detach and be replaced by new ones. A freezing lemma such as this for TA has the potential to resolve open problems in established models. Consider the Signal Tile Assembly Model (STAM), where signals are "fire-once", i.e., not reusable. A generalized variant allowing perpetual reuse of signals would plausibly yield substantial power and ease system design. This freezing result in TA will give us the first tool needed in proving the conjecture that single-fire STAM is just as powerful as the perpetual-fire STAM.

2   Model and Definitions

A Tile Automata system is a marriage between cellular automata and 2-handed self-assembly. Systems consist of a set of monomer tile states, along with local affinities between states denoting the strength of attraction between adjacent monomer tiles in those states. A set of local state-change rules are included for pairs of adjacent states. Assemblies (collections of edge-connected tiles) in the model are created from an initial set of starting assemblies by combining previously built assemblies given sufficient binding strength from the affinity function. Further, existing assemblies may change states of internal monomer tiles according to any applicable state change rules. An example system is shown in Figure 1.

2.1   States, tiles, and assemblies

Tiles and States. Consider an alphabet of *state types*[4] Σ. A tile t is an axis-aligned unit square centered at a point $L(t) \in \mathbb{Z}^2$. Further, tiles are assigned a state type from Σ, where $S(t)$ denotes the state type for a given tile t. We say two tiles t_1 and t_2 are of the same *tile type* if $S(t_1) = S(t_2)$.

Affinity Function. An *affinity function* takes as input an element in $\Sigma^2 \times D$, where $D = \{\bot, \vdash\}$, and outputs an element in $\mathbb{N}$. This output is referred to as the *affinity strength* between two states, given direction $d \in D$. Directions $\bot$ and $\vdash$ indicate above-below and side-by-side orientations of states, respectively.

Transition Rules. Transition rules allow states to change based on their neighbors. Formally, a *transition rule* is a 5-tuple $(S_{1a}, S_{2a}, S_{1b}, S_{2b}, d)$ with each $S_{1a}, S_{2a}, S_{1b}, S_{2b} \in \Sigma$ and $d \in D = \{\bot, \vdash\}$. Essentially, a transition rule says that if states S_{1a} and S_{2a} are adjacent to each other, with a given orientation d, they can transition to states S_{1b} and S_{2b} respectively.

Assemblies. A *positioned shape* is any subset of $\mathbb{Z}^2$. A *positioned assembly* is a set of tiles at unique coordinates in $\mathbb{Z}^2$, and the positioned shape of a positioned assembly $\mathcal{A}$ is the set of coordinates of those tiles, denoted as $\texttt{SHAPE}_{\mathcal{A}}$. For a positioned assembly $\mathcal{A}$, let $\mathcal{A}(x, y)$ denote the state type of the tile with location $(x, y) \in \mathbb{Z}^2$ in $\mathcal{A}$.

---

[4] We note that Σ does not include an "empty" state. In tile self-assembly, unlike cellular automata, positions in $\mathbb{Z}^2$ may have no tile (and thus no state).

For a given positioned assembly $\mathcal{A}$ and affinity function Π, define the *bond graph* $G_{\mathcal{A}}$ to be the weighted grid graph in which:

- each tile of $\mathcal{A}$ is a vertex,
- no edge exists between non-adjacent tiles,
- the weight of an edge between adjacent tiles T_1 and T_2 with locations (x_1, y_1) and (x_2, y_2), respectively, is
  - $\Pi(S(T_1), S(T_2), \perp)$ if $y_1 > y_2$,
  - $\Pi(S(T_2), S(T_1), \perp)$ if $y_1 < y_2$,
  - $\Pi(S(T_1), S(T_2), \vdash)$ if $x_1 < x_2$,
  - $\Pi(S(T_2), S(T_1), \vdash)$ if $x_1 > x_2$.

A positioned assembly $\mathcal{A}$ is said to be τ-*stable* for positive integer τ provided the bond graph $G_{\mathcal{A}}$ has min-cut at least τ.

For a positioned assembly $\mathcal{A}$ and integer vector $\boldsymbol{v} = (v_1, v_2)$, let $\mathcal{A}_{\boldsymbol{v}}$ denote the positioned assembly obtained by translating each tile in $\mathcal{A}$ by vector $\boldsymbol{v}$. An *assembly* is a set of all translations $\mathcal{A}_{\boldsymbol{v}}$ of a positioned assembly $\mathcal{A}$. A *shape* is the set of all integer translations for some subset of $\mathbb{Z}^2$, and the shape of an assembly A is defined to be the set of the positioned shapes of all positioned assemblies in A. The *size* of either an assembly or shape X, denoted as $|X|$, refers to the number of elements of any positioned assembly of X.

Breakable Assemblies. An assembly is τ-*breakable* if it can be split into two assemblies along a cut whose total affinity strength sums to less than τ. Formally, an assembly C is *breakable* into assemblies A and B if the bond graph $G_{\mathcal{C}}$ for some positioned assembly $\mathcal{C} \in C$ has a cut $(\mathcal{A}, \mathcal{B})$ for positioned assemblies $\mathcal{A} \in A$ and $\mathcal{B} \in B$ of affinity strength less than τ. We call assemblies A and B *pieces* of the breakable assembly C.

Combinable Assemblies. Two assemblies are τ-*combinable* provided they may attach along a border whose strength sums to at least τ. Formally, two assemblies A and B are τ-*combinable* into an assembly C provided $G_{\mathcal{C}}$ for any $\mathcal{C} \in C$ has a cut $(\mathcal{A}, \mathcal{B})$ of strength at least τ for some positioned assemblies $\mathcal{A} \in A$ and $\mathcal{B} \in B$. We call C a *combination* of A and B.

Transitionable Assemblies. Consider some set of transition rules Δ. An assembly A is *transitionable*, with respect to Δ, into assembly B if and only if there exist $\mathcal{A} \in A$ and $\mathcal{B} \in B$ such that for some pair of adjacent tiles $t_i, t_j \in \mathcal{A}$:

- $\exists$ a pair of adjacent tiles $t_h, t_k \in \mathcal{B}$ with $L(t_i) = L(t_h)$ and $L(t_j) = L(t_k)$
- $\exists$ a transition rule $\delta \in \Delta$ s.t. $\delta = (S(t_i), S(t_j), S(t_h), S(t_k), \perp)$ or
  $\delta = (S(t_i), S(t_j), S(t_h), S(t_k), \vdash)$
- $\mathcal{A} - \{t_i, t_j\} = \mathcal{B} - \{t_h, t_k\}$

2.2 Tile Automata model (TA)

A *tile automata system* is a 5-tuple $(\Sigma, \Pi, \Lambda, \Delta, \tau)$ where Σ is an alphabet of state types, Π is an affinity function, Λ is a set of initial assemblies with each tile assigned a state from Σ, Δ is a set of transition rules for states in Σ, and $\tau \in \mathbb{N}$ is the *stability threshold*. When the affinity function and state types are implied, let (Λ, Δ, τ) denote a tile automata system. An example tile automata system can be seen in Figure 1.

(a) Tile Automata System Γ.　　　(b) The producibles and terminals of Γ.
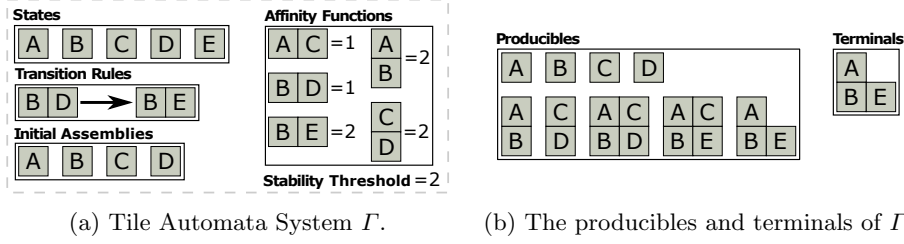
Fig. 1: An example of a tile automata system Γ. Recursively applying the transition rules and affinity functions to the initial assemblies of a system yields a set of producible assemblies. Any producibles that cannot combine with, break into, or transition to another assembly are considered to be terminal.

Definition 1 (Tile Automata Producibility). *For a given tile automata system $\Gamma = (\Sigma, \Lambda, \Pi, \Delta, \tau)$, the set of* producible assemblies *of Γ, denoted $\mathtt{PROD}_\Gamma$, is defined recursively:*

- *(Base) $\Lambda \subseteq \mathtt{PROD}_\Gamma$*
- *(Recursion) Any of the following:*
  - *(Combinations) For any $A, B \in \mathtt{PROD}_\Gamma$ such that A and B are τ-combinable into C, then $C \in \mathtt{PROD}_\Gamma$.*
  - *(Breaks) For any $C \in \mathtt{PROD}_\Gamma$ such that C is τ-breakable into A and B, then $A, B \in \mathtt{PROD}_\Gamma$.*
  - *(Transitions) For any $A \in \mathtt{PROD}_\Gamma$ such that A is transitionable into B (with respect to Δ), then $B \in \mathtt{PROD}_\Gamma$.*

For a system $\Gamma = (\Sigma, \Lambda, \Pi, \Delta, \tau)$, we say $A \rightarrow_1^\Gamma B$ for assemblies A and B if A is τ-combinable with some producible assembly to form B, if A is transitionable into B (with respect to Δ), if A is τ-breakable into assembly B and some other assembly, or if $A = B$. Intuitively this means that A may grow into assembly B through one or fewer combinations, transitions, and breaks. We define the relation $\rightarrow^\Gamma$ to be the transitive closure of $\rightarrow_1^\Gamma$, ie., $A \rightarrow^\Gamma B$ means that A may grow into B through a sequence of combinations, transitions, and/or breaks.
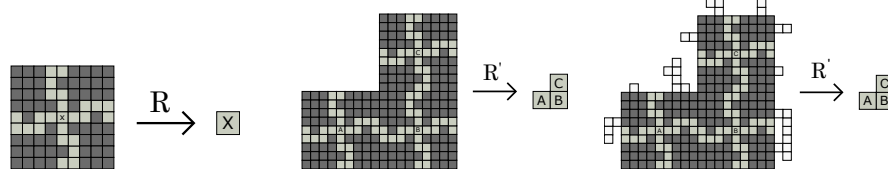
Definition 2 (Terminal Assemblies). *A producible assembly A of a tile automata system $\Gamma = (\Sigma, \Lambda, \Pi, \Delta, \tau)$ is* terminal *provided A is not τ-combinable with any producible assembly of Γ, A is not τ-breakable, and A is not transitionable to any producible assembly of Γ. Let $\mathtt{TERM}_\Gamma \subseteq \mathtt{PROD}_\Gamma$ denote the set of producible assemblies of Γ which are terminal.*

Definition 3 (Unique Assembly). *A tile automata system Γ* uniquely produces *an assembly A if $A \in \mathtt{TERM}_\Gamma$ and for all $B \in \mathtt{PROD}_\Gamma$, $B \rightarrow^\Gamma A$.*

Definition 4 (Unique Shape Assembly). *A tile automata system Γ* uniquely assembles *a shape S provided that for all $A \in \mathtt{PROD}_\Gamma$, there exists some $B \in \mathtt{TERM}_\Gamma$ of shape S such that $A \rightarrow^\Gamma B$.*

Definition 5 (Freezing). *Consider a tile automata system $\Gamma = (\Sigma, \Lambda, \Pi, \Delta, \tau)$ and a directed graph G constructed as follows:*

- *each state type $\sigma \in \Sigma$ is a vertex*

(a) An example entry in R: an m-block representation function with $m = 9$. (b) An example entry in R': a positioned assembly replacement function. (c) An example entry in R': the same replacement function with c-Fuzz.

Fig. 2: Examples of m-block representation and mapping. (a) Essentially, the partial function R, called an m-block representation function, takes a macro-block and maps it to a state in the state space of some other system. (b) The function R' takes a positioned assembly, containing m-blocks, and maps it to a positioned assembly over the state space of the other system using the m-block representation function to perform the mapping. (c) The lighter tiles represent c-fuzz which does not change the mapping of the macro-block.

– *for any two state types* $\alpha, \beta \in \Sigma$, *an edge from* α *to* β *exists if and only if there exists a transition rule in* Δ *s.t.* α *transitions to* β

Γ *is said to be* freezing *if* G *is acyclic and* non-freezing *otherwise. Intuitively, a tile automata system is freezing if any one tile in the system can never return to a state which it held previously. This implies that any given tile in the system can only undergo a finite number of state transitions.*

2.3 Simulation Definitions

In this subsection we formally define what it means for one tile automata system to *simulate* another. We use a standard block-representation scheme, similar to what is done in [5], in which the simulating system maps $m \times m$ blocks of states (for a scale factor m simulation) to single states within the simulated system's state space. With this block mapping we can generate an *assembly* mapping as shown in Figure 2. A system is said to *simulate* another system at scale factor m if such a block mapping exists such that it follows the rules laid out in this section. The purpose of these rules is to provide a reasonable definition for simulating the dynamics of a particular system. More exhaustive definitions for simulation have been considered before (see [3]); however, our intent is to provide relatively straightforward rules that allow for some flexibility while still capturing the essence of what it means for one system to simulate another.

Consider two tile automata systems Γ and Γ'. Let Σ_Γ and $\Sigma_{\Gamma'}$ denote the set of state types used in Γ and Γ', respectively.

Macro-blocks and assemblies. An *m-block assembly*, or *macro-block*, is a partial function $\lambda : \mathbb{Z}_m \times \mathbb{Z}_m \to \Sigma_\Gamma$, where $\mathbb{Z}_m = \{0, 1, \ldots, m-1\}$. Let $B_m^{\Sigma_\Gamma}$ be the set of all m-block assemblies over Σ_Γ. The m-block with no domain of definition is said to be *empty*.

For an arbitrary positioned assembly $\mathcal{A}$ over state space Σ_Γ, define $\mathcal{A}_{x,y}^m$ to be the m-block defined by $\mathcal{A}_{x,y}^m(i, j) = \mathcal{A}(mx + i, my + j)$ for $0 \le i, j < m$.

Macro-block representation and mapping. As demonstrated in Figure 2, our simulation definition uses a macro-block representation and mapping scheme. For a partial function $R : B_m^{\Sigma_\Gamma} \to \Sigma_{\Gamma'}$, known as an *m-block representation function*, define the partial function R' that takes as input a positioned assembly $\mathcal{A}$ over state space Σ_Γ and outputs a positioned assembly over state space $\Sigma_{\Gamma'}$. With T denoting a function whose input is an element in $\Sigma \times \mathbb{Z}^2$ and $T(\sigma, x, y)$ outputting a tile with state σ and location (x, y), define $R'(\mathcal{A}) = \{T(R(\mathcal{A}_{x,y}^m), x, y) \mid$ for all non-empty blocks $\mathcal{A}_{x,y}^m$ s.t. $\mathcal{A}_{x,y}^m \in dom(R)\}$.

*c***-Fuzz.** The concept of c-fuzz is essentially the idea that a macro-block can have a bounded number of "extra" tiles attached to it without altering its mapping. This allows a simulating system to make minor intermediate attachments while enacting the simulation. Another way to think of c-fuzz is as a reasonable allowance for limited-size non-empty macro-blocks (that map to an empty tile in the simulated system) to be used in the simulation process. Formally, a mapping $R'(\mathcal{A}) = \mathcal{A}'$ is said to have *c-fuzz*, for some constant c, if and only if for all non-empty blocks $\mathcal{A}_{x,y}^m$, it is the case that $(x + u, y + v) \in dom(\mathcal{A}')$ for some $u, v \in [-c, c]$. R' is said to have c-fuzz if and only if every such mapping $R'(\mathcal{A}) = \mathcal{A}'$ has c-fuzz for all $\mathcal{A} \in dom(R')$. R has c-fuzz if R' has c-fuzz.

Assembly Replacement. For a c-fuzz R', define the *assembly replacement function* $R^* : \text{PROD}_\Gamma \to \text{PROD}_{\Gamma'}$ such that $R^*(A) = A'$ if and only if there exists a positioned assembly $\mathcal{A} \in A$ s.t. $R'(\mathcal{A}) \in A'$. When discussing the application of R^* to a set of assemblies Υ, we use the notation $R^*(\Upsilon)$, where $R^*(\Upsilon) = \{R^*(A) | A \in \Upsilon\}$.

Validity. A c-fuzz assembly replacement $R^*(A)$ is called *valid* if and only if: (1) $R'(\mathcal{A}) = \mathcal{A}', \forall \mathcal{A} \in A$, or (2) $R'(\mathcal{A}) = \varnothing$ and the minimum-diameter bounding square of $\mathcal{A}$ is $\leq 2mc, \forall \mathcal{A} \in A$.

The assembly replacement function R^* is said to be *Γ-valid* if $R^*(A)$ is valid for all $A \in \Gamma$. The m-block representation function R is said to be Γ-valid if and only if R^* is Γ-valid.

Simulation. Given a tile automata system Γ, a tile automata system Γ', a constant c, and a Γ-valid c-fuzz m-block representation function $R : B_m^{\Sigma_\Gamma} \to \Sigma_{\Gamma'}$ we say Γ *simulates* tile automata system Γ' *under the c-fuzz rule* if and only if:

- $R^*(\text{PROD}_\Gamma) \supset \Lambda_{\Gamma'}$.
- For any two assemblies $A, B \in \text{PROD}_\Gamma$ s.t. $R^*(A) = \varnothing$ and $R^*(B) = \varnothing$, A and B can combine to form C only if the following is true:
  - $R^*(C) = \varnothing$
  - or, $R^*(C) \in \Lambda_{\Gamma'}$
- For any two assemblies $A', B' \in \text{PROD}_{\Gamma'}$, the following is true:
  - if $A' \to^{\Gamma'} B'$ then it must be that $\exists A, B \in \text{PROD}_\Gamma$ s.t. $R^*(A) = A'$, $R^*(B) = B'$, and $A \to^\Gamma B$.
  - if $A' \not\to^{\Gamma'} B'$ then it must be that $\forall A, B \in \text{PROD}_\Gamma$ where $R^*(A) = A'$, and $R^*(B) = B'$, $A \not\to^\Gamma B$.
- $\forall A \in \text{TERM}_\Gamma$: if $R^*(A) = A' \in \text{PROD}_{\Gamma'}$, then it must also be that $A' \in \text{TERM}_{\Gamma'}$.

Observation. It is important to note that with $R^*(\text{PROD}_\Gamma) \supset \Lambda_{\Gamma'}$, it follows directly from the application of our dynamics simulation definitions that $R^*(\text{PROD}_\Gamma) = \text{PROD}_{\Gamma'}$.
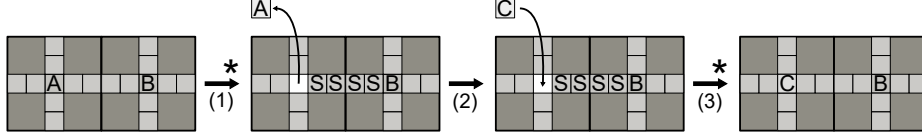
Fig. 3: A simplified overview of simulating the transition $AB \vdash CB$. $\xrightarrow{*}$ indicates a sequence of combination, breaking, and/or transition events occurring. The middle tile in the blocks are the clock tiles, and the rest are wires. Before (1) occurs, the blocks are attached at the adjacent wire tiles with affinity $\Pi(A, B, \vdash)$ from the original system. During (1), a signal proceeds down the wire from B to A. Once the signal reaches A, A detaches. (2) is the attachment event where C is placed within the formerly-A block. (3) indicates a signal returning down the wire after the C block has finished its transition. During (3), when the signal passes back through the boundary between the blocks, tiles are left where the wires meet with affinities matching $\Pi(A, B, \vdash)$, allowing combination/breaking events to follow matching the original system.

3 Simulating Non-freezing with Freezing Tile Automata

Here, we present the main result of the paper: for any non-freezing TA system, there is a freezing TA system that simulates it. Subsection 3.1 gives an overview of the construction. Subsection 3.2 presents some primitives for the construction. Subsection 4 gives a formal statement of the theorem and its proof.

3.1 Simulation Overview

At a high-level, the approach is to simulate state transitions between tiles with a process whereby a tile detaches from the assembly and is replaced by a new tile. In this way, any cyclic state transitions are simulated by instead detaching the tile whose state is to be transitioned and attaching a tile with the new state in its place. One immediate issue with a naïve, scale-1 version of this approach is connectivity— e.g., in a 1×3 assembly, replacing the middle tile while keeping the assembly connected is non-trivial.

This issue motivates using a *block* scheme wherein each tile in the original system is simulated by a larger square block of tiles. The larger scale factor allows blocks to stay connected while some interior tiles detach and are replaced. In the center of the blocks is a *clock tile*, which determines which tile in the original system the block maps to in the m-block representation function. Extending from the clock tile to the four edges of the block are *wires*, a connected path of tiles which (1) send information via token-passing to adjacent blocks about initiating state transitions and (2) attach to wires on other blocks with affinities corresponding to the original system.

A high-level overview omitting some particular details follows. Attachment and detachment events are simulated by the wires exposing affinities matching that of the original system. To simulate state transitions, several steps occur. A simplified summary is in Figure 3. It begins with a sequence of state transitions, called *signals*, beginning from a clock tile proceeding down a wire to an adjacent block's clock tile. Upon receiving signals from all neighbors, the clock

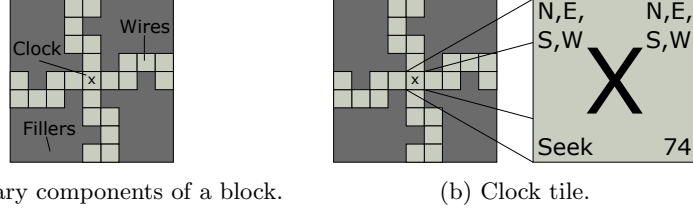(a) Primary components of a block. (b) Clock tile.

Fig. 4: Blocks and clock tiles. (a) The primary components of a block are the filler tiles (used for connectivity), the wire tiles (used for the passing of signals), and the clock tile (used to control signal flow). (b) The clock tile contains information about which of its tokens (N,E,S,W) it has, which of its neighbors tokens (N,E,S,W) it has, which mode it is in (seeking, sending, off), and how many of the constant number of transitions have occurred.

tile detaches from the block, and a new clock tile representing the new state of the block takes its place. The wires are designed to be *replaceable*; in some cases, while sending a sequence of state transitions down the wire, the wire tiles detach (one-by-one) and are replaced with new tiles. This alleviates the issue of the wires themselves dissatisfying the freezing constraint. When a signal passes the boundary of one block and enters another, these tiles have full τ-strength affinity. This ensures the tiles may not detach while the transition occurs. After the clock tile is replaced and the signal passes back through this boundary, it leaves a tile with affinity matching the post-transition tile in the original system.

3.2 Simulation Primitives

Blocks. One block is constructed for each of the initial tile types in the original system. Each block consists of 3 portions; *filler* tiles, *wire* tiles, and a *clock* tile. The filler tiles are simply needed for the block to maintain connectivity when replacing the clock or portions of the wire. The filler tiles undergo no state transitions, save for one during the initial assembling of the block. The wires are responsible for propagating a block's incoming and outgoing signals to initiate transition rules between blocks' clocks. The wires in a block are used to maintain the affinities between blocks (all affinity between two blocks is between wire tiles) The clock tiles are the middle tile of each block, and send/receive signals to/from the wire tiles which can initiate a state transition of that clock or another clock. The clock tile is the main determinant used in the m-block replacement function (discussed in the simulation definition). A clock tile is designed to *represent* exactly one state x of the system to be simulated. We label the clock tiles' states according to the state they represent. We say a block *represents* exactly one state x if its clock tile represents x.

Wires. Since each tile of the original system is replaced by a block, *wires* send transition rules from the middle of the block to the edges. Wires send a cascade of transition rules along a path of connected tiles. As an example (Fig. 5), given a path of horizontally connected w tiles and the transition rule $w_r w \vdash w_r w_r$, if the leftmost tile is transitioned to w_r (e.g. by a tile x to its left and the rule $xw \vdash xw_r$), the transition cascades down the path of w tiles (and, e.g., transitioning a tile y to z at the end of the wire by the rule $w_r y \vdash w_r z$.
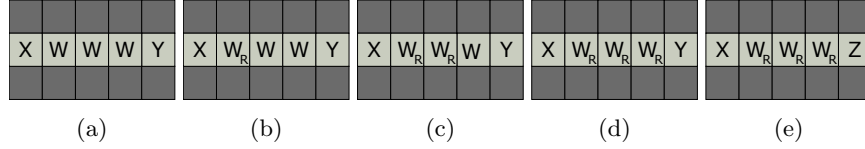
Fig. 5: A wire demonstrating its signal-passing ability. Given the rules $XW \vdash XW_R$, $W_R W \vdash W_R W_R$, and $W_R Y \vdash W_R Z$ we can see the signal propogation from (a) to (e).
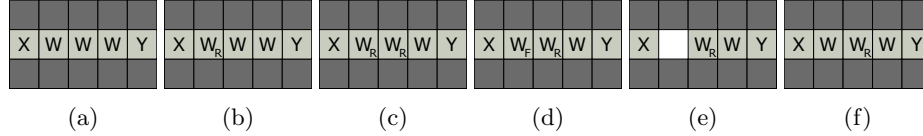

Fig. 6: A wire replacing tiles while passing a signal. Given the rules $W_R W \vdash W_R W_R$ and $W_R W_R \vdash W_F W_R$ we replace tiles once they transition. Once a signal has started propagating in (a) with the rule $XW \vdash XW_R$ yielding (b), any further transitions with allow (d) to occur. The tile with W_F has no affinity to its neighbors so it detaches (e) and a new tile with state W attaches (f).

Then, the presence of the x tile has been detected by the non-adjacent y tile using a series of transitions along the wire.

In order to reuse the wires, the tiles must be replaced after at most a constant number of uses due to freezing transition rules; otherwise, the wire could be reset with transition rules alone. Figure 6 depicts this signal passing with the required tile replacements. Towards a wire with replaceable tiles, consider the following transition rules: $w_r w \vdash w_r w_r$ and $w_r w_r \vdash w_f w_r$. The first rule passes the transition along the wire. The second rule sets the previous tile to a "fall off" state which is not bound to the assembly which contains the wire. The w_f tile may detach from the assembly, and a new w tile may attach in its place.

State-state Wires. We augment the wire scheme with *state-state* information stored via the wire tiles' states. State-state refers to the two states represented by the clock tiles which the wire lies between. When a block representing x is first constructed, and when its wire is only touching one clock since the block has no neighbor in that direction, the wires' state-state is referred to as x-$\varnothing$. Without loss of generalization, x-$\varnothing$ is the information on the wire protruding east, and $\varnothing$-x is the information on the wire protruding west. If the wire is between two clocks, perhaps representing x and y, the state-state is then referred to as x-y (if the x block is to the west of the y block). When a block representing x has a new adjacent neighboring block representing y via an attachment to another assembly or via a block-state transition (Section 3.2), the state-state wire between the two blocks must be updated. For example, the result may be an x-z wire meeting a w-y wire. Via state transition, the wire information should then be updated to x-y. On a horizontal state-state wire (without loss of generalization), this updating is done by the following rules: if two state-state wires disagree, e.g. an x-z wire tile on the left meets a w-y wire tile on the right, the x-z wire changes the w-y wire tile to x-y. Similarly, the w-y wire tile can change

the x-z tile to x-y. This works since the tile on the left-hand wire tile has the correct information about the left-hand block, and the right-hand wire tile has the right information about the right-hand block.

Seeking, Sending, and Off States. Clocks transition between *seeking*, *sending*, and *off* states depending on their adjacent state-state wire information. Transition of a clock tile representing x to the *seeking* state may occur if and only if an adjacent x-y wire is present such that a transition rule exists (w.r.t. the cardinal direction that the wire is coming from) between x and y that changes x to another state. Transition of a clock tile representing x to the sending state may occur if and only if a neighbor block may transition to the seeking state. Explicitly, this transition can occur if and only if an adjacent x-y wire is present such that a transition rule exists (w.r.t. the cardinal direction that the wire is coming from) between x and y that changes y to another state. Transition of a clock tile to the off state may occur if and only if a clock holds all of its own *tokens* (tokens will be described in the next paragraph) and no others. The off state of the clock halts all token passing by the block. The purpose of these states is to simulate terminal assemblies. Assemblies with no possible state transitions are simulated by blocks which are all in the off state, halting transitions through the wires. If neighboring blocks have no applicable transition rules, then the seeking/sending state cannot be reached.

Token Passing. *Tokens* are passed between neighboring blocks using the wire signal passing scheme shown earlier. Clock tiles are responsible for sending and receiving tokens. A clock tile can have up to eight tokens: four of its own tokens (one for each cardinal direction) and up to four of its neighbor's tokens (one for each cardinal direction). If a clock is in the seeking or sending state, it may send its token through the wire to the clock on the other end. Token ownership is represented by the state of the clock tile. The following rules hold for token passing:

- With respect to one cardinal direction, tiles can have: their own token, their own token and their neighbor token, or no tokens, i.e., a clock cannot have its neighbor token but not its own. This is enforced via clock transition rules wherein clocks cannot send their token if they hold their neighbor's token.
- Tokens cannot pass through each other on the wire; if two tokens meet on the wire, one is (nondeterministically) forced back to its clock.
- Tiles in the off state cannot receive tokens.

Block-state Transitions. When a clock receives all eight possible tokens (its four own tokens and its four neighbor tokens), the clock may undergo a *block-state transition*: a series of transitions within the clock's block which changes the state in the simulated system which the block represents. The clock, upon receiving its eighth token, may go through the following sequence which simulates a state transition: First, the clock undergoes a transition due to one of its neighboring state-state wires (which inform the clock of what states his neighbor blocks represent). This way, the clock nondeterministically samples from the state transitions it may simulate based on the represented state of its neighbor blocks. Once selecting a state to transition to, the clock stores (in an adjacent
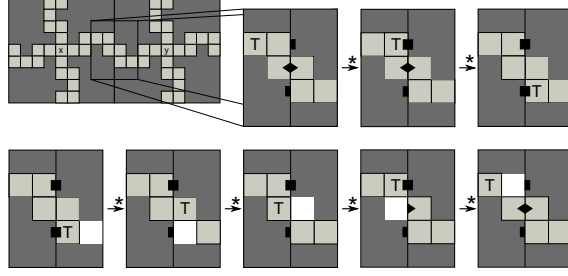
Fig. 7: Token passing between two blocks representing x and y. Squares on the edge of blocks signify τ-strength affinity. Rhombuses signify affinities equal to the affinity between x and y in the simulated system. As before, $\xrightarrow{*}$ indicates a sequence of attachment, detachment, and/or combination events have occurred. In the top sequence of transitions, the x block passes its token to y. As the token passes the border between the two blocks, the states in the wire bind with τ strength with the other block to ensure the blocks cannot detach until the token is returned. In the bottom sequence of transitions, the y block sends x's token back. In this case, the τ strength affinities with each block are removed, and only an affinity matching that of the state to be simulated remains. As the token returns, each wire tile is replaced with new tiles.

wire tile) information about that state. Then, the clock tile transitions to a state in which it has no affinities and detaches from the block. A new tile attaches in its place whose state is designed to read from the adjacent wire which stored the information about which state it will become from the previous clock tile. Once the new clock tile's state is updated with the previous clock's information, the wire tile which stored the information then undergoes a state transition and detaches to be replaced with a new wire tile. Then, the clock tile updates its adjacent state-state wires to a new state-state wire effectively overwriting the old state from the wire and replacing with the new state (e.g., an x-y wire becomes a z-y wire as the block simulates a transition from x to z).

Clock Replacement. As the clocks send and receive tokens, they undergo state transitions. Therefore, the clock tiles must be replaced after a finite number of token passes. Each clock has a counter which increments each time it undergoes a state transition. Once the clock reaches an arbitrarily designated value, the clock will undergo a replacement. The clock first stores (in an adjacent wire tile) information about its possessed tokens and the state in the simulated system which it represents. Then, the clock tile transitions to a state in which it detaches from the block. A new tile attaches in its place whose state is designed to read from the adjacent wire which stored the information from the previous clock tile. Once the new clock tile's state is updated with the previous clock's information, the wire tile which stored the information then undergoes a state transition and detaches to be replaced with a new wire tile.

Dummy Blocks. To initiate a block-state transition (Section 3.2), a block requires four neighbors. Of course, in the simulated system, not all transitionable tiles will have neighboring tiles. To alleviate this, include a set of blocks called *dummy blocks.* Dummy blocks act as temporary neighbors to blocks which lack
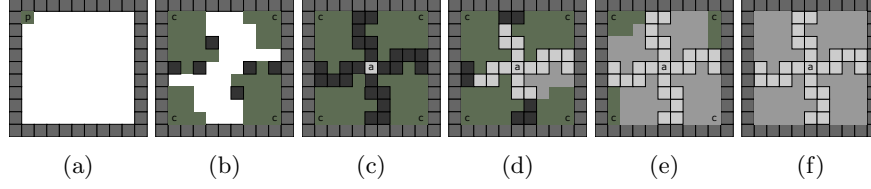
Fig. 8: Block construction process. (a) The block construction process begins with a pre-assembled frame. Four construction initiator tiles attach to the corners of the frame, initiating the assembling of the pre-filler tiles. (b) Once each of the pre-filler portions of the block are complete, blank wire tiles can begin attaching. (c) Upon completion of all four wire portions, a seed-clock tile can attach and (d) begin changing the blank wires into wires of the same type as the clock. (e) When a wire segment has been changed to a typed-wire, it begins transitioning the pre-filler tiles into filler tiles. (f) When all four filler sections have transitioned, the block no longer has any affinity with the frame, and detaches.

them. Dummy blocks may pass tokens to neighboring blocks, but cannot receive them. Include one set of dummy tiles for each cardinal direction. Dummy blocks have two states: attach and detach. Dummy blocks in the attach state may attach to any block in the system with τ strength from one direction, e.g., the north dummy block binds its south edge to the north edge of any block in the system. Include a state transition between any block and the dummy block which transitions the dummy block from its attach state to its detach state.

The detach state has no affinity with any blocks in the system except in the case that a neighbor has received the dummy block's token, in which case a full τ strength bond is held. Then, dummy blocks may attach to unoccupied positions in the assembly, and subsequently transition and detach; however, they may first pass a token, in which case they are attached to the assembly until the token is removed from the neighbor. In this way, any block in the assembly with a missing neighbor has a chance at attaching a dummy block neighbor and grabbing its token. Dummy blocks cannot attach to blocks which are off.

3.3 Additional Simulation Primitives

Here we further detail a few of the primitives used in our construction with details that are not as important, but are useful nonetheless.

Wire Replacement. As discussed prior, wires may replace tiles as signals are passing through. Wires replace their tiles under the following circumstances: 1) if the block's neighbor's token is being sent back to its neighbor, and 2) if the block's own token is returning. Otherwise, signals may pass through the wires freely. These two conditions enforce that the wire is replaced after a finite number of signals are passed through.

Exposed Affinities on Blocks. The following rules are imposed on the affinities of the wire tiles of a block which are exposed to other blocks:

– If the block's token has not passed through the wire (the block still has its token), and the neighbor's token has not passed through the wire (the block does not have its neighbor's token), the affinity exposed matches the affinity exposed by the state in the to-be-simulated system that the block represents.

– Otherwise, as a token passes through the wire causing the above condition to fail, the wire attaches with full τ strength to the neighboring block.

These rules ensure that all detachment and attachment events of the to-be-simulated system may occur by the blocks, since the affinities exposed by blocks match those of the simulated states when the tokens meet the above requirements. Additionally, these rules ensure that when a block is undergoing a state transition, the block is attached with τ strength to his neighbors to ensure a detachment does not occur prior to the state change. The process whereby the affinity changes on the wire during token-passing is shown in Figure 7.

Block Construction. The blocks must be constructed by a series of combination events beginning with single tiles. To imitate the tiles of the original system, the blocks must use one tile on each edge to expose the affinities of the original tile. Moreover, these edge affinities must be exposed "all at once" in order to simulate the behavior of the original tiles (i.e., incomplete blocks may expose only the northbound affinity of the original tile, whereas the original tiles expose all of their affinities from the get-go). To achieve this, the blocks are constructed inside a *frame*, inhibiting their affinities from being exposed. Then, a transition rule occurs between the block and the frame indicating that the block has completed construction, in which the frame detaches from the block. This process can be seen in Figure 8.

4 Simulation Proof

Theorem 1. *Given a tile automata system Γ', there exists a freezing tile automata system Γ which simulates Γ' under the 2-fuzz rule via a 9-block replacement function.*

Proof. For a given tile automata system $\Gamma' = (\Sigma', \Lambda', \Pi', \Delta', \tau')$, we generate a tile automata system $\Gamma = (\Sigma, \Lambda, \Pi, \Delta, \tau)$. In Σ, include seed, clock, and wire tile types representing each state type in Σ'. Further, include the $\mathcal{O}(1)$ state types required for the block construction and dummy blocks (Sec. 3.2).

Stability Threshold. Γ requires $\tau \geq 2$ to use the wire technique 3.2. If Γ' has $\tau' = 1$, Γ must have $\tau = 2$. In this case, affinities of strength 1 in Γ' are simulated by affinities of strength 2 in Γ when blocks expose affinities on the wire designed to match the original system. Otherwise, to simulate a system with $\tau' \geq 2$, the simulating system uses $\tau = \tau'$.

State Complexity ($|\Sigma|$). Σ (the set of state types of Γ) includes state-state wires (Section 3.2) for each pair of states in Σ'. Due to this, $|\Sigma| = \mathcal{O}(|\Sigma'|^2)$. All other techniques require at most $c * \Sigma'$ state types for some constant c.

The Macro-block Representation and Mapping. The mapping of macro-blocks in Γ to states in Γ' is straightforward: for a state $s \in \Gamma'$, there exists a block in Γ whose clock represents s. Any block containing the clock representing s is mapped to s in the macro-block representation function R. When the clock is detached from the block, either through a block-state transition (Sec. 3.2) or a clock replacement (Sec. 3.2), the block is mapped according to the neighboring wire tile which is used to temporarily store the information of the clock tile.

2-Fuzz Rule. As the blocks of Γ' are being constructed via the block construction process, the clock tile does not represent any state in Γ. These blocks, along with the frame they are assembled within, still satisfy simulation under the 2-fuzz rule (diameter of the minimum-diameter bounding square of the blocks with frame is $< 2mc$), and hence map to the empty assembly. Additionally, the attachment of dummy blocks (Section 3.2) which do not map to any states in Γ are also permissible under the 2-fuzz rule.

Initial Assemblies Our construction is designed such that for every tile in each of the initial assemblies of Γ', there exists a block in Γ that was produced via the block construction process described above. Thus, we see that $R^*(\texttt{PROD}_\Gamma) \supset \Lambda_{\Gamma'}$.

Simulating Dynamics: Part 1. Consider the assemblies $A', B' \in \texttt{PROD}_{\Gamma'}$ s.t. $A' \to^{\Gamma'} B'$. Suppose that A' can transition into B' via an attachment using state s. Any assembly $A \in \texttt{PROD}_\Gamma$ where $R^*(A) = A'$ contains a 9×9 block which represents s. The A whose 9×9 "s"-block clock only has all four of its own tokens (and is not currently attached to a dummy block) is guaranteed to be able to make the same attachments via its 9×9 "s"-block as A' is via s. Now, suppose that A' can transition into B' via a state-transition of s into s'. Again, any assembly $A \in \texttt{PROD}_\Gamma$ where $R^*(A) = A'$ contains a 9×9 block which represents s. The A whose 9×9 "s"-block clock has collected all four of its own tokens, and all four of its neighbors' tokens is guaranteed to be able to make the same transitions via its 9×9 "s"-block as A' is via s.

Simulating Dynamics: Part 2. Consider the assemblies $C, D \in \texttt{PROD}_\Gamma$. Suppose that $C \to^\Gamma D$. There are only a few instances where $R^*(C) \neq R^*(D)$. First, note that none of the internal state-transitions of the 9×9 blocks that make up C, which are required for token-passing, alter the mapping of C. Nor do the attachment of dummy blocks to C alter its mapping. So for all of these transitions, $R^*(C) = R^*(D)$. So, the only instances $R^*(C) \neq R^*(D)$ would be due to a "block-sized" detachment event, an attachment event involving C and some other assembly, or a block-state transition within C. Since each 9×9 block in C inherits its affinity from the states in $R^*(C)$, any "block-sized" attachment/detachment events which involve C could only occur if their state-equivalent events were possible in $R^*(C)$. Furthermore, since block-state transitions are inherited the same way, the only block-state transitions that could occur in C must also be driven by equivalent events that could occur in $R^*(C)$.

Simulating Dynamics: Part 3. Consider an assembly $E \in \texttt{TERM}_\Gamma$. We know that every exposed wire on the perimeter of E must not have affinity towards any other 9×9 block in the system. This can only occur if $R^*(E)$ cannot attach to any other assembly in Γ'. Also, every clock in E must be stuck in the off state, meaning no transitions are possible. This can only occur if $R^*(E)$ cannot transition into any other assembly in Γ' via a state transition. It must also be the case that E is not breakable into any other assemblies. Since all of the clocks in A are off, we know that, internally, each 9×9 block is not breakable. Furthermore, we know that each 9×9 block in E is bound to its neighbors with a total strength of at least τ. This can only occur if $R^*(E)$ is not breakable. Therefore, by definition, $R^*(E) \in \texttt{TERM}_{\Gamma'}$. □

5   Conclusion & Future Work

This work introduces Tile Automata as a hybrid between tile self-assembly and cellular automata. The model resembles other more complicated, well-studied forms of active self-assembly, and thus results about simulation between TA and other active self-assembly models should be pursued. We have shown in this work that freezing TA can simulate non-freezing TA, allowing future proofs about general TA to apply to freezing systems. Some optimizations are open: the simulation herein uses 9×9 macro-blocks and a quadratic state-complexity increase to achieve non-freezing behavior with a freezing system; a smaller macro-block size and smaller state-complexity increase are welcome.

References

1. Florent Becker, Diego Maldonado, Nicolas Ollinger, and Guillaume Theyssier. Universality in freezing cellular automata. *CoRR*, abs/1805.00059, 2018. Available from: `http://arxiv.org/abs/1805.00059`.
2. Sarah Cannon, Erik D. Demaine, Martin L. Demaine, Sarah Eisenstat, Matthew J. Patitz, Robert Schweller, Scott M. Summers, and Andrew Winslow. Two hands are better than one (up to constant factors): Self-assembly in the 2ham vs. atam. In *STACS*, volume 20 of *LIPIcs*, pages 172–184, 2013.
3. Erik D. Demaine, Matthew J. Patitz, Trent A. Rogers, Robert T. Schweller, Scott M. Summers, and Damien Woods. The two-handed tile assembly model is not intrinsically universal. *Algorithmica*, 74(2):812–850, Feb 2016.
4. Zahra Derakhshandeh, Robert Gmyr, Thim Strothmann, Rida Bazzi, Andréa W. Richa, and Christian Scheideler. *Leader Election and Shape Formation with Self-organizing Programmable Matter*, pages 117–132. Springer Inter. Publishing, 2015.
5. David Doty, Jack H. Lutz, Matthew J. Patitz, Robert Schweller, Scott M. Summers, and Damien Woods. The tile assembly model is intrinsically universal. In *Proc. of the 53rd Conf. on Foundations of Computer Science*, FOCS'12, 2012.
6. Eric Goles, Diego Maldonado, Pedro Montealegre, and Nicolas Ollinger. On the computational complexity of the freezing non-strict majority automata. In *Int. Workshop on Cellular Automata and Discrete Complex Sys.*, pages 109–119, 2017.
7. Eric Goles, Nicolas Ollinger, and Guillaume Theyssier. Introducing freezing cellular automata. In *Cellular Automata and Discrete Complex Systems*, volume 24 of *AUTOMATA '15*, pages 65–73, 2015.
8. Natasa Jonaska and Daria Karpenko. Active tile self-assembly, part 2: Self-similar structures and structural recursion. *J. of Fou. of Com. Sci.*, 25(02):165–194, 2014.
9. Jarkko Kari. Theory of cellular automata: A survey. *Theoretical Computer Science*, 334(1):3 – 33, 2005.
10. Jennifer E. Padilla, Matthew J. Patitz, Raul Pena, Robert T. Schweller, Nadrian C. Seeman, Robert Sheline, Scott M. Summers, and Xingsi Zhong. Asynchronous signal passing for tile self-assembly: Fuel efficient computation and efficient assembly of shapes. *Inter. Journal of Foundations of Computer Science*, 25:459, 2014.
11. Jennifer E. Padilla, Ruojie Sha, Martin Kristiansen, Junghuei Chen, Natasha Jonoska, and Nadrian C. Seeman. A signal-passing dna-strand-exchange mechanism for active self-assembly of dna nanostructures. *Angewandte Chemie International Edition*, 54(20):5939–5942, 2015.
12. Damien Woods, Ho-Lin Chen, Scott Goodfriend, Nadine Dabby, Erik Winfree, and Peng Yin. Active self-assembly of algorithmic shapes and patterns in polylogarithmic time. In *Innov. in Theor. Comp. Sci.*, ITCS'13, pages 353–354, 2013.